Prometheus & Grafana sur Kubernetes — Installation et configuration

Guide complet pour déployer et configurer une stack de supervision Prometheus + Grafana sur un cluster Kubernetes via kube-prometheus, sans Helm. Dashboards, alertes, PromQL, notifications email Mailjet et accès Grafana.

60 min de lecture **Niveau Intermédiaire**

Document généré le 25/06/2026 à 21h38 · nouvy.fr/wiki/prometheus-grafana-kubernetes

Sommaire

39 section(s) · 60 min de lecture

1. Présentation

Pourquoi une stack de supervision dédiée au cluster ?

Un cluster Kubernetes génère une quantité importante de métriques : état des pods, consommation CPU/RAM par namespace, santé des nœuds, disponibilité des déploiements... Ces métriques ne sont pas visibles dans une stack Docker classique.

kube-prometheus est la solution officielle de la communauté Prometheus pour superviser un cluster Kubernetes. Elle déploie tous les composants nécessaires via des **manifests YAML**, sans Helm.

Composants installés

Composant	Rôle
Prometheus	Collecte les métriques du cluster et des nœuds
kube-state-metrics	Expose l'état des objets K8s (pods, deployments, services, PVC...)
node-exporter	Métriques système de chaque nœud (CPU, RAM, disque)
Alertmanager	Routage et envoi des alertes (email, Slack...)
Grafana	Dashboards préconfigurés pour Kubernetes

> *Tout s'installe dans le namespace **monitoring**, créé automatiquement.*

---

2. Prérequis

- Cluster Kubernetes opérationnel (k3s, kubeadm, EKS, GKE, AKS...)
- `kubectl` installé et configuré sur ta machine (`kubectl get nodes` doit répondre)
- `git` installé

Vérifier que kubectl est fonctionnel

```
kubectl get nodes
# NAME          STATUS   ROLES           AGE   VERSION
# k8s-master    Ready    control-plane   10d   v1.28.0
# k8s-worker1   Ready    <none>          10d   v1.28.0
```

Copier

---

3. Installation de kube-prometheus

Cloner le repo officiel

```
git clone https://github.com/prometheus-operator/kube-prometheus.git
cd kube-prometheus
```

Copier

> *Le repo contient déjà tous les manifests YAML prêts à l'emploi dans le dossier* `manifests/`*.*

Installer en deux étapes

```
# Étape 1 — CRDs et namespace monitoring (obligatoire en premier)
kubectl apply --server-side -f manifests/setup/

# Attendre que les CRDs soient prêts
kubectl wait \
  --for condition=Established \
  --all CustomResourceDefinition \
  --namespace=monitoring

# Étape 2 — Stack complète (Prometheus, Grafana, Alertmanager, exporters)
kubectl apply -f manifests/
```

Copier

> `--server-side` *est requis pour l'étape setup car les CRDs sont trop volumineux pour un apply client-side classique.*

Vérifier que tout est Running

```
kubectl get pods -n monitoring
# NAME                                  READY   STATUS
# prometheus-k8s-0                      2/2     Running
# prometheus-k8s-1                      2/2     Running  ← 2 réplicas par défaut
# alertmanager-main-0                   2/2     Running
# grafana-xxx                           1/1     Running
# kube-state-metrics-xxx                3/3     Running
# node-exporter-xxx                     2/2     Running  ← un pod par nœud
# prometheus-operator-xxx               2/2     Running
# blackbox-exporter-xxx                 2/2     Running
```

Copier

```
# Vérifier aussi les services exposés
kubectl get svc -n monitoring
```

Copier

---

4. Accéder à Grafana

Accès temporaire (port-forward)

```
kubectl port-forward -n monitoring svc/grafana 3000:3000
# → http://localhost:3000
# Login : admin / admin
```

Copier

> *Le port-forward fonctionne uniquement pendant que la commande tourne. Couper avec `Ctrl+C`.*

Accès permanent via NodePort

Sans Ingress Controller, la solution la plus simple est d'exposer Grafana via un **NodePort** — le service devient accessible directement sur l'IP d'un nœud du cluster.

```
# grafana-nodeport.yml
apiVersion: v1
kind: Service
metadata:
  name: grafana-nodeport
  namespace: monitoring
spec:
  type: NodePort
  selector:
    app.kubernetes.io/name: grafana
  ports:
    - port: 3000
      targetPort: 3000
      nodePort: 30300   # port accessible sur chaque nœud (range 30000-32767)
```

Copier

```
kubectl apply -f grafana-nodeport.yml

# Récupérer l'IP d'un nœud
kubectl get nodes -o wide
# → http://<IP_NOEUD>:30300
```

Copier

> *Le NodePort est accessible sur **chaque nœud** du cluster, pas seulement sur le master.*

---

5. Accéder à Prometheus et Alertmanager

```
# Prometheus
kubectl port-forward -n monitoring svc/prometheus-k8s 9090:9090
# → http://localhost:9090

# Alertmanager
kubectl port-forward -n monitoring svc/alertmanager-main 9093:9093
# → http://localhost:9093
```

Copier

---

6. Dashboards Grafana pour Kubernetes

Dans Grafana → **Dashboards → Import** → entrer l'ID :

ID	Dashboard	Utilisation
7249	Kubernetes Cluster Monitoring	Vue globale du cluster
315	Kubernetes Cluster (générale)	CPU, RAM, pods par nœud
6417	Kubernetes Pods	Détail par pod
1860	Node Exporter Full	Métriques système des nœuds

> *Les dashboards sont aussi **déjà préconfigurés** dans Grafana après l'installation de kube-prometheus. Aller dans **Dashboards → Browse → Kubernetes**.*

---

7. Métriques PromQL utiles

État du cluster

```
# Nœuds non Ready
kube_node_status_condition{condition="Ready", status!="true"}

# Pods non Running (hors Completed)
kube_pod_status_phase{phase!="Running", phase!="Succeeded"}

# Pods en erreur (CrashLoopBackOff, OOMKilled…)
kube_pod_container_status_waiting_reason{reason=~"CrashLoopBackOff|OOMKilled|Error"}
```

Copier

CPU et RAM

```
# CPU utilisé par namespace
sum(rate(container_cpu_usage_seconds_total{container!=""}[5m])) by (namespace)

# RAM utilisée par pod
sum(container_memory_working_set_bytes{container!=""}) by (pod, namespace)

# RAM utilisée vs limite définie
container_memory_working_set_bytes / container_spec_memory_limit_bytes
```

Copier

Déploiements et réplicas

```
# Réplicas disponibles vs souhaités
kube_deployment_status_replicas_available / kube_deployment_spec_replicas

# Déploiements avec réplicas insuffisants
kube_deployment_status_replicas_available < kube_deployment_spec_replicas
```

Copier

Réseau

```
# Trafic réseau entrant par pod (Mo/s)
rate(container_network_receive_bytes_total[5m]) / 1024 / 1024

# Trafic réseau sortant par pod (Mo/s)
rate(container_network_transmit_bytes_total[5m]) / 1024 / 1024
```

Copier

---

8. Alertes préconfigurées

kube-prometheus installe automatiquement des règles d'alerte couvrant :

Alerte	Déclenchement
`KubeNodeNotReady`	Nœud non Ready depuis > 15 min
`KubePodCrashLooping`	Pod en CrashLoopBackOff
`KubeDeploymentReplicasMismatch`	Réplicas insuffisants
`KubeMemoryOvercommit`	RAM demandée > RAM disponible
`KubeCPUOvercommit`	CPU demandé > CPU disponible
`PrometheusTargetMissing`	Une cible de scrape a disparu

Voir toutes les règles :

```
kubectl get prometheusrule -n monitoring
```

Copier

Tester le déclenchement d'une alerte

Pour vérifier que la chaîne complète fonctionne (Prometheus → Alertmanager → Email), créer un pod qui crash :

```
# Crée un pod qui crash en boucle → déclenche KubePodCrashLooping après ~15 min
kubectl run crashloop --image=busybox -- sh -c "exit 1"

# Surveiller le pod
kubectl get pod crashloop -w

# Supprimer après le test
kubectl delete pod crashloop
```

Copier

On peut aussi injecter une alerte manuellement via `amtool` :

```
kubectl exec -n monitoring alertmanager-main-0 -c alertmanager -- \
  amtool alert add TestEmail severity=critical instance=test \
  --alertmanager.url=http://localhost:9093
```

Copier

---

9. Configurer les notifications email (Alertmanager + Mailjet)

Étape 1 — Créer le Secret pour le mot de passe SMTP

Ne pas mettre le mot de passe SMTP en clair dans la config Alertmanager. Utiliser un Secret Kubernetes dédié :

```
kubectl create secret generic alertmanager-mailjet-secret \
  -n monitoring \
  --from-literal=smtp-password='MAILJET_SECRET_KEY'
```

Copier

Étape 2 — Créer la config Alertmanager

Créer le fichier `alertmanager.yaml` :

```
global:
  smtp_smarthost: "in-v3.mailjet.com:587"
  smtp_from: "alertes@mondomaine.com"
  smtp_auth_username: "MAILJET_API_KEY"
  smtp_auth_password_file: /etc/alertmanager/secrets/alertmanager-mailjet-secret/smtp-
password
  smtp_require_tls: true

route:
  receiver: "equipe-si"
  group_by: ["alertname", "instance"]
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 4h
  routes:
    - match:
        severity: critical
      receiver: "equipe-si-critical"

receivers:
  - name: "equipe-si"
    email_configs:
      - to: "admin@mondomaine.com"
        send_resolved: true
        headers:
          Subject: "[SUPERVISION] {{ .GroupLabels.alertname }} — {{ .Status | toUpper }}"

  - name: "equipe-si-critical"
    email_configs:
      - to: "admin@mondomaine.com"
        send_resolved: true
        headers:
          Subject: "[CRITIQUE] {{ .GroupLabels.alertname }} — {{ .Status | toUpper }}"

  - name: "null"

inhibit_rules:
  - source_match:
      severity: critical
    target_match:
      severity: warning
    equal: ["alertname", "instance"]
```

Copier

Étape 3 — Appliquer le Secret Alertmanager

```
kubectl create secret generic alertmanager-main \
  -n monitoring \
  --from-file=alertmanager.yaml \
  --dry-run=client -o yaml | kubectl apply -f -
```

Copier

> *Alertmanager recharge la config automatiquement (hot-reload).*

Étape 4 — Monter le Secret SMTP dans Alertmanager

Patcher le CRD Alertmanager pour monter le secret contenant le mot de passe :

```
# alertmanager-patch.yaml
apiVersion: monitoring.coreos.com/v1
kind: Alertmanager
metadata:
  name: main
  namespace: monitoring
spec:
  volumes:
    - name: mailjet-secret
      secret:
        secretName: alertmanager-mailjet-secret
  volumeMounts:
    - name: mailjet-secret
      mountPath: /etc/alertmanager/secrets/alertmanager-mailjet-secret
      readOnly: true
```

Copier

```
kubectl apply -f alertmanager-patch.yaml
```

Copier

Étape 5 — Vérifier

```
# Vérifier que le secret est monté
kubectl exec -n monitoring alertmanager-main-0 -c alertmanager -- \
  cat /etc/alertmanager/secrets/alertmanager-mailjet-secret/smtp-password

# Voir les logs
kubectl logs -n monitoring alertmanager-main-0 -c alertmanager --tail=30

# Vérifier la config chargée via l'UI
kubectl port-forward -n monitoring svc/alertmanager-main 9093:9093
# → http://localhost:9093 → onglet Status
```

Copier

Envoyer une alerte de test pour confirmer que l'email part :

```
kubectl exec -n monitoring alertmanager-main-0 -c alertmanager -- \
  amtool alert add TestEmail severity=critical instance=test \
  --alertmanager.url=http://localhost:9093
```

Copier

Un email doit arriver sur la boîte du receiver dans les ~30s (délai `group_wait`).

Erreur fréquente : "cannot read smtp password from file"

Si les logs affichent ce genre d'erreur au chargement de la config :

```
msg="Loading configuration file failed" ... err="cannot read smtp password from file"
```

Copier

C'est que l'**étape 2 (monter le secret) n'a pas été faite** ou que le pod n'a pas redémarré

après.

Fix en une ligne (le plus simple) :

```
kubectl patch alertmanager main -n monitoring --type merge \
  -p '{"spec":{"secrets":["alertmanager-mailjet-secret"]}}'
```

Copier

Cette commande ajoute (ou remplace) le montage du secret via `spec.secrets`. Le prometheus-operator redémarre automatiquement le pod Alertmanager, et le fichier `smtp-password` devient disponible à `/etc/alertmanager/secrets/alertmanager-mailjet-secret/smtp-password`.

Vérifier ensuite :

```
# Attendre le redémarrage
kubectl rollout status statefulset/alertmanager-main -n monitoring

# Vérifier que le fichier est bien monté
kubectl exec -n monitoring alertmanager-main-0 -c alertmanager -- \
  cat /etc/alertmanager/secrets/alertmanager-mailjet-secret/smtp-password

# Vérifier les logs (plus d'erreur de chargement)
kubectl logs -n monitoring alertmanager-main-0 -c alertmanager --tail=20
```

Copier

> *Si le secret `alertmanager-mailjet-secret` lui-même n'existe pas, il faut d'abord le créer avec la commande de l'étape 1.*

Cas Gmail

Utiliser un **App Password** (pas le mot de passe du compte) :

```
global:
  smtp_smarthost: "smtp.gmail.com:587"
  smtp_from: "toi@gmail.com"
  smtp_auth_username: "toi@gmail.com"
  smtp_auth_password: "app-password-16-chars"
  smtp_require_tls: true
```

Copier

Cas Microsoft 365

SMTP AUTH doit être activé sur le tenant :

```
global:
  smtp_smarthost: "smtp.office365.com:587"
  smtp_from: "alerts@tondomaine.fr"
  smtp_auth_username: "alerts@tondomaine.fr"
  smtp_auth_password: "MOT_DE_PASSE"
  smtp_require_tls: true
```

Copier

---

10. Troubleshooting — Alertmanager n'envoie pas les emails

Symptômes

- La config Alertmanager est correcte (vérifiée dans l'UI `/status`)
- Les credentials SMTP sont bons (testés en direct, ex. via un pod Python)
- Le secret contenant le mot de passe est bien monté dans le pod
- DNS + port SMTP (587) joignables depuis le pod
- **Aucune erreur dans les logs Alertmanager**
- **Aucun `Notify success` non plus** — l'envoi ne semble simplement jamais se déclencher
- `amtool alert add` fonctionne (alerte bien visible dans Alertmanager) mais pas d'email

Cause

Le pod Alertmanager peut avoir un **token de Service Account devenu invalide** (après redémarrage du cluster, rotation des clés SA, ou problème de signature). Le pod tourne, monte bien ses secrets, mais son état interne l'empêche d'émettre les notifications. Exactement le même symptôme qu'avec `kube-proxy` quand son token expire — il continue à tourner sans jamais appliquer les règles.

Fix

Redémarrer simplement le pod Alertmanager :

```
kubectl rollout restart statefulset/alertmanager-main -n monitoring
# ou
kubectl delete pod -n monitoring alertmanager-main-0
```

Copier

Le nouveau pod récupère un token frais et les emails partent immédiatement après la prochaine alerte.

Vérification après fix

Activer temporairement le log debug pour voir passer les notifications :

```
kubectl patch alertmanager main -n monitoring --type merge \
  -p '{"spec":{"logLevel":"debug"}}'

# Envoyer une alerte de test
kubectl exec -n monitoring alertmanager-main-0 -c alertmanager -- \
  amtool alert add TestEmail severity=critical instance=test \
  --alertmanager.url=http://localhost:9093

# Attendre ~35s (group_wait) puis vérifier les logs
kubectl logs -n monitoring alertmanager-main-0 -c alertmanager | grep -i notify
# → doit contenir : level=DEBUG msg="Notify success" ... integration=email[0]

# Remettre le log level normal
kubectl patch alertmanager main -n monitoring --type merge \
  -p '{"spec":{"logLevel":"info"}}'
```

Copier

> *Alertmanager **ne log pas les envois réussis au niveau `info`**. Sans debug, l'absence de log ne veut pas dire que ça ne marche pas — il faut forcer le debug pour confirmer.*

Bonus — Même problème sur kube-proxy

Le même symptôme peut toucher `kube-proxy` après un long uptime ou une rotation de clés SA : les Services NodePort deviennent inaccessibles (aucune règle iptables/nftables), et les logs du pod affichent :

```
failed to list *v1.Service: Unauthorized
```

Copier

Fix identique :

```
kubectl -n kube-system rollout restart daemonset kube-proxy
```

Copier

---

11. Mettre à jour la stack

```
cd kube-prometheus
git pull

# Mettre à jour les CRDs
kubectl apply --server-side -f manifests/setup/

# Mettre à jour la stack
kubectl apply -f manifests/
```

Copier

> *`kubectl apply` est **idempotent** : seuls les composants modifiés sont mis à jour, les autres ne bougent pas.*

12. Désinstaller

```
cd kube-prometheus
kubectl delete -f manifests/
kubectl delete -f manifests/setup/
```

Copier

Récapitulatif — Ordre de mise en place

Étape	Action	Commande
1	Vérifier kubectl	`kubectl get nodes`
2	Cloner kube-prometheus	`git clone https://github.com/prometheus-operator/kube-prometheus.git`
3	Installer CRDs	`kubectl apply --server-side -f manifests/setup/`
4	Installer la stack	`kubectl apply -f manifests/`
5	Vérifier les pods	`kubectl get pods -n monitoring`
6	Accéder à Grafana	NodePort 30300 ou `kubectl port-forward svc/grafana 3000:3000`
7	Importer dashboards	ID 7249, 315, 6417, 1860
8	Créer le secret SMTP	`kubectl create secret generic alertmanager-mailjet-secret ...`
9	Configurer Alertmanager	Secret `alertmanager-main` + patch CRD
10	Tester les alertes	`kubectl run crashloop --image=busybox -- sh -c "exit 1"`
11	Si pas d'email → restart	`kubectl rollout restart statefulset/alertmanager-main -n monitoring`